

Python: module cdms.tvariable

cdms.tvariable

[index](#)

TransientVariable (created by createVariable)
is a child of both AbstractVariable and the masked array class.
Contains also the write part of the old cu interface.

Modules

[MA](#)

[Numeric](#)

[PropertiedClasses](#)

[types](#)

Classes

MA.MA.MaskedArray(builtin .object)
TransientVariable(cdms.avariable.AbstractVariable, MA.MA.MaskedArray)
TransientVariable(cdms.avariable.AbstractVariable, MA.MA.MaskedArray)
cdms.avariable.AbstractVariable(cdms.cdmsobj.CdmsObj, cdms.slabinterface.Slab)
TransientVariable(cdms.avariable.AbstractVariable, MA.MA.MaskedArray)
TransientVariable(cdms.avariable.AbstractVariable, MA.MA.MaskedArray)

class **TransientVariable**(cdms.avariable.AbstractVariable, MA.MA.MaskedArray)

An in-memory variable.

Method resolution order:

TransientVariable
cdms.avariable.AbstractVariable
cdms.cdmsobj.CdmsObj
cdms.internattr.InternalAttributesClass
PropertiedClasses.Properties.PropertiedClass
cdms.slabinterface.Slab
MA.MA.MaskedArray
builtin .object

Methods defined here:

__init__(self, data, typecode=None, copy=0, savespace=0, mask=None, fill_value=None, grid=None, id=None, copyaxes=1)

createVariable (self, data, typecode=None, copy=0, savespace=0, mask=None, fill_value=None, grid=None, axes=None, attributes=None, id=None)

data is the data or a *tuple* giving the shape (not a list!)

```
__len__(self)
    Length of first dimension

__repr__(self)

__str__(self)

assignValue(self, data)

astype(self, tc)
    return self as array of given type.

clone(self, copyData=1)
    clone (self, copyData=1)
    Return a copy of self as a transient variable.
    If copyData is 1 (default), make a separate copy of the data.

copyAxis(self, n, axis)
    Set n axis of self to a copy of axis. (0-based index)
    Invalidates grid.

copyDomain(self, other)
    Set the axes and grid by copying variable other.

copydimension(self, idim, other, jdim)
    Set idim dimension of self to variable other's jdim'th
    This is for old cu compatibility. Use copyAxis for new code.

expertSlice(self, slicelist)

getAxis(self, n)

getDomain(self)

getGrid(self)

getValue(self, squeeze=1)

initDomain(self, axes, copyaxes=1)

isEncoded(self)
    Transient variables are not encoded

setAxis(self, n, axis, savegrid=0)
    Set n axis of self to a copy of axis. (0-based index)

setAxisList(self, axislist)
    Set the axes to axislist.

setMaskFromGridMask(self, mask, gridindices)
    Set the mask for self, given a grid mask and the variable dom
    indices corresponding to the grid dimensions.
```

```
setMissing(self, value)
    Set missing value attribute and fill value

set_fill_value(self, value)
    Set missing value attribute and fill value

setdimattribute(self, dim, field, value)
    Set the attribute named field from the dim'th dimension.
```

typecode(self)

Data and other attributes defined here:

count = 0

Methods inherited from cdms.avariable.AbstractVariable:

```
__abs__(self)

__add__(self, other)

__array__(self, t=None)

__call__(self, *args, **kwargs)
    Selection of a subregion using selectors

__div__(self, other)

__eq__(self, other)

__ge__(self, other)

__getitem__(self, key)

__getslice__(self, low, high)

__gt__(self, other)

__iadd__(self, other)
    Add other to self in place.

__idiv__(self, other)
    Divide self by other in place.

__imul__(self, other)
    Multiply self by other in place.

__isub__(self, other)
    Subtract other from self in place.

__le__(self, other)
```

```

lshift(self, n)
lt(self, other)
mul(self, other)
ne(self, other)
neg(self)
pow(self, other, third=None)
radd = add(self, other)
rdiv(self, other)
rmul = mul(self, other)
rshift(self, n)
rsub(self, other)
sqr(self)
sub(self, other)

crossSectionRegrid(self, newLevel, newLatitude, missing=None, order=None, method='log')
    Return the variable regridded to new pressure levels and latitudes.
    The variable should be a function of lat, level, and (optional) time.
    <newLevel> is an axis of the result pressure levels.
    <newLatitude> is an axis of latitude values.
    <method> is optional, either "log" to interpolate in the log
        or "linear" for linear interpolation.
    <missing> and <order> are as for regrid.CrossSectionRegridder.

decode(self, ar)
    Decode compressed data. ar is a masked array, scalar, or MA masked array.

generateGridkey(self, convention, vardict)
    generateGridkey(): Determine if the variable is gridded,
    and generate ((latname, lonname, order, maskname, class), latname)
    or (None, None, None) if not gridded. vardict is the variable's
    grid key.

generateRectGridkey(self, lat, lon)
    generateRectGridkey(): Determine if the variable is gridded,
    and generate (latname, lonname, order, maskname, class) if gridded
    or None if not gridded

getAxisIds(self)
    Get a list of axis identifiers

getAxisIndex(self, axis_spec)

```

Return the index of the axis specified by axis_spec.
Argument axis_spec and be as for axisMatches
Return -1 if no match.

getAxisList(self, axes=None, omit=None, order=None)

Get the list of axis objects;
If axes is not None, include only certain axes.
If omit is not None, omit those specified by omit.
Arguments omit or axes may be as specified in axisMatchAxis
order is an optional string determining the output order

getAxisListIndex(self, axes=None, omit=None, order=None)

Return a list of indices of axis objects;
If axes is not None, include only certain axes.
less the ones specified in omit. If axes is None,
use all axes of this variable.
Other specifications are as for axisMatchIndex.

getConvention(self)

Get the metadata convention associated with this object.

getGridIndices(self)

Return a tuple of indices corresponding to the variable grid.

getLatitude(self)

Get the first latitude dimension, or None if not found.

getLevel(self)

Get the first vertical level dimension in the domain,
or None if not found.

getLongitude(self)

Get the first longitude dimension, or None if not found.

getMissing(self, asarray=0)

Return the missing value as a scalar, or as
a Numeric array if asarray==1

getOrder(self, ids=0)

getOrder(ids=0) returns the order string, such as tzyx.

if ids == 0 (the default) for an axis that is not t,z,x,y
the order string will contain a '-' in that location.
The result string will be of the same length as the number
of axes. This makes it easy to loop over the dimensions.

if ids == 1 those axes will be represented in the order
string as (id) where id is that axis' id. The result will
be suitable for passing to order2index to get the
corresponding axes, and to orderparse for dividing up into
components.

```

getRegion(self, *specs, **keys)
    getRegion
        Read a region of data. A region is an n-dimensional
        rectangular region specified in coordinate space.
        'slices' is an argument list, each item of which has one of the
        following forms:
        - x, where x is a scalar
            Map the scalar to the index of the closest coordinate value.
        - (x,y)
            Map the half-open coordinate interval [x,y) to index interval
        - (x,y,'cc')
            Map the closed interval [x,y] to index interval. Other options
            are 'oc' (open on the left), and 'co' (open on the right, the
            default).
        - (x,y,'co',cycle)
            Map the coordinate interval with wraparound. If no cycle is
            specified, it will occur iff axis.isCircular() is true.
            NOTE: Only one dimension may be wrapped.
        - Ellipsis
            Represents the full range of all dimensions bracketed by
            ellipses.
        - ':' or None
            Represents the full range of one dimension.

```

For example, suppose the variable domain is (time,level,lat,lon).

```
getRegion((10,20),850,Ellipsis,(-180,180))
```

retrieves:

- all times t such that $10 \leq t < 20$.
- level 850
- all values of all dimensions between level and lon (namely lat)
- longitudes x such that $-180 \leq x < 180$. This will be wrapped around if lon.topology=='linear'

getSlice(self, *specs, **keys)

x.getSlice takes arguments of the following forms and produces a return array. The keyword argument squeeze determines whether or not the shape of the returned array contains dimensions whose length is 1; by default this argument is 1, and such dimensions are 'squeezed out'.

There can be zero or more positional arguments, each of the following forms:

- (a) a single integer n, meaning slice(n, n+1)
- (b) an instance of the slice class
- (c) a tuple, which will be used as arguments to create a slice
- (d) None or ':', which means a slice covering that entire dimension
- (e) Ellipsis (...), which means to fill the slice list with 'Ellipsis' leaving only enough room at the end for the remaining positional arguments

There can be keyword arguments of the form key = value, where key can be one of the names 'time', 'level', 'latitude', or 'longitude'. The corresponding value can be any of (a)-(d) above.

There must be no conflict between the positional arguments and the keywords.

In (a) – (c) negative numbers are treated as offsets from the end of that dimension, as in normal Python indexing.

***getTime*(self)**

Get the first time dimension, or None if not found

***isAbstractCoordinate*(self)**

***pressureRegrid*(self, newLevel, missing=None, order=None, method='log')**

Return the variable regridded to new pressure levels.

The variable should be a function of lat, lon, pressure, and <newLevel> is an axis of the result pressure levels.

<method> is optional, either "log" to interpolate in the log or "linear" for linear interpolation.

<missing> and <order> are as for regrid.PressureRegridder.

***rank*(self)**

***reg_specs2slices*(self, initSpecList, force=None)**

***regrid*(self, togrid, missing=None, order=None, mask=None)**

return self regridded to the new grid. Keyword arguments are as for regrid.Regridder.

***reorder*(self, order)**

return self reordered per the specification order

***select = __call__*(self, *args, **kwargs)**

Selection of a subregion using selectors

***setGrid*(self, grid)**

Set the variable grid

***specs2slices*(self, specList, force=None)**

Create an equivalent list of slices from an index specification.

An index specification is a list of acceptable items, which are:

-- an integer

-- a slice instance (slice(start, stop, stride))

-- the object "unspecified"

-- the object None

-- a colon

The size of the specList must be rank()

***subRegion*(self, *specs, **keys)**

***subSlice*(self, *specs, **keys)**

Methods inherited from cdms.cdmsobj.CdmsObj:

***dump*(self, path=None, format=1)**

```

dump(self, path=None, format=1)
    Dump an XML representation of this object to a file.
    'path' is the result file name, None for standard output.
    'format'==1 iff the file is formatted with newlines for readability.

matchPattern(self, pattern, attribute, tag)
    # Match a pattern in a string-valued attribute. If attribute
    # search all string attributes. If tag is not None, it must match

matchone(self, pattern, attrname)
    # Return true iff the attribute with name attrname is a string
    # attribute which matches the compiled regular expression pattern
    # if attrname is None and pattern matches at least one string
    # attribute. Return false if the attribute is not found or is not a string.

searchPattern(self, pattern, attribute, tag)
    # Search for a pattern in a string-valued attribute. If attribute
    # search all string attributes. If tag is not None, it must match

searchPredicate(self, predicate, tag)
    # Apply a truth-valued predicate. Return a list containing a string
    # if the predicate is true and either tag is None or matches
    # If the predicate returns false, return an empty list

searchone(self, pattern, attrname)
    Return true iff the attribute with name attrname is a string
    attribute which contains the compiled regular expression pattern
    if attrname is None and pattern matches at least one string
    attribute. Return false if the attribute is not found or is not a string.

```

Methods inherited from [cdms.internattr.InternalAttributesClass](#):

```

is_internal_attribute(self, name)
    is internal attribute(name) is true if name is internal.

replace_external_attributes(self, newAttributes)
    replace external attributes(newAttributes)
    Replace the external attributes with dictionary newAttributes

```

Methods inherited from [PropertiedClasses.Properties.PropertiedClass](#):

```

__delattr__(self, name)

__getattr__(self, name)

__setattr__(self, name, value)

get_property_d(self, name)
    Return the 'del' property handler for name that self uses.
    Returns None if no handler.

```

```
get_property_g(self, name)
    Return the 'get' property handler for name that self uses.
    Returns None if no handler.

get_property_s(self, name)
    Return the 'set' property handler for name that self uses.
    Returns None if no handler.

set_property(self, name, actg=None, acts=None, actd=None, nowrite=None, nodelete=None)
    Set attribute handlers for name to methods actg, acts, actd
    None means no change for that action.
    nowrite = 1 prevents setting this attribute.
    nowrite defaults to 0.
    nodelete = 1 prevents deleting this attribute.
    nodelete defaults to 1 unless actd given.
    if nowrite and nodelete is None: nodelete = 1
```

Methods inherited from [cdms.slabinterface.Slab](#):

```
createattribute(self, name, value)
    Create an attribute and set its name to value.

deleteattribute(self, name)
    Delete the named attribute.

getattribute(self, name)
    Get the attribute name.

getdimattribute(self, dim, field)
    Get the attribute named field from the dim'th dimension.
    For bounds returns the old cu one-dimensional version.

info(self, flag=None, device=None)
    Write info about slab; include dimension values and weights if

listall(self, all=None)
    Get list of info about this slab.

listattributes(self)
    Return a list of attribute names.

listdimattributes(self, dim)
    List the legal axis field names.

listdimnames(self)
    Return a list of the names of the dimensions.

setattribute(self, name, value)
    Set the attribute name to value.

showdim(self)
    Show the dimension attributes and values.
```

Data and other attributes inherited from [cdms.slabinterface.Slab](#):

`std_slab_atts` = ['filename', 'missing_value', 'comments', 'grid_name', 'grid_type', 'time_statistic', 'lo'

Methods inherited from [MA.MA.MaskedArray](#):

`__and__(self, other)`

Return bitwise_and

`__float__(self)`

Convert self to float.

`__floordiv__(self, other)`

Return divide(self, other)

`__int__(self)`

Convert self to int.

`__mod__(self, other)`

Return remainder(self, other)

`__or__(self, other)`

Return bitwise_or

`__pos__(self)`

Return array(self)

`__rand__ = __and__(self, other)`

Return bitwise_and

`__rfloordiv__(self, other)`

Return divide(other, self)

`__rmod__(self, other)`

Return remainder(other, self)

`__ror__ = __or__(self, other)`

Return bitwise_or

`__rtruediv__(self, other)`

Return divide(other, self)

`__rxor__ = __xor__(self, other)`

Return bitwise_xor

`__setitem__(self, index, value)`

Set item described by index. If value is masked, mask those 1

`__setslice__(self, i, j, value)`

Set slice i:j; if value is masked, mask those locations.

```
__truediv__(self, other)
    Return divide(self, other)

__xor__(self, other)
    Return bitwise_xor

byte_swapped(self)
    Returns the raw data field, byte_swapped. Included for consistency
    with Numeric but doesn't make sense in this context.

compressed(self)
    A 1-D array of all the non-masked data.

dot(self, other)
    s.dot(other) = innerproduct(s, other)

fill_value(self)
    Get the current fill value.

filled(self, fill_value=None)
    A Numeric array with masked values filled. If fill_value is None
    use fill_value().

    If mask is None, copy data only if not contiguous.
    Result is always a contiguous, Numeric array.

ids(self)
    Return the ids of the data and mask areas

iscontiguous(self)
    Is the data contiguous?

itemsize(self)
    Item size of each data item.

mask(self)
    Return the data mask, or None. Result contiguous.

outer(self, other)
    s.outer(other) = outerproduct(s, other)

put(self, values)
    Set the non-masked entries of self to filled(values).
    No change to mask

putmask(self, values)
    Set the masked entries of self to filled(values).
    Mask changed to None.

raw_data(self)
    The raw data; portions may be meaningless.
    May be noncontiguous. Expert use only.
```

***raw_mask*(self)**
The raw mask; portions may be meaningless.
May be noncontiguous. Expert use only.

***savespace*(self, value)**
Set the spacesaver attribute to value

***size*(self, axis=None)**
Number of elements in array, or in a particular axis.

***spacesaver*(self)**
spacesaver() queries the spacesaver attribute.

***tolist*(self, fill_value=None)**
Convert to list

***tostring*(self, fill_value=None)**
Convert to string

***unmask*(self)**
Replace the mask by None if possible.

***unshare_mask*(self)**
If currently sharing mask, make a copy.

Properties inherited from MA.MA.MaskedArray:

flat
Access array in flat form.
get">get = _get_flat(self)
Calculate the flat value.
set">set = _set_flat(self, value)
x.flat = value

imag
Access the imaginary part of the array
get">get = _get_imaginary(self)
Get the imaginary part of a complex array.
set">set = _set_imaginary(self, value)
x.imaginary = value

imaginary
Access the imaginary part of the array
get">get = _get_imaginary(self)
Get the imaginary part of a complex array.
set">set = _set_imaginary(self, value)
x.imaginary = value

real
Access the real part of the array
get">get = _get_real(self)
Get the real part of a complex array.

```

set">set = _set_real(self, value)
    x.real = value

shape
tuple giving the shape of the array
get">get = _get_shape(self)
    Return the current shape.
set">set = _set_shape(self, newshape)
    Set the array's shape.

```

Data and other attributes inherited from [MA.MA.MaskedArray](#):

```

__dict__ = <dictproxy object>
    dictionary for instance variables (if defined)

__weakref__ = <attribute '__weakref__' of 'MaskedArray' objects>
    list of weak references to the object (if defined)

handler_cache_key = 'MA.MaskedArray'

```

createVariable = class TransientVariable([cdms.avariable.AbstractVariable](#), [MA.MA.MaskedArray](#))

An in-memory variable.

Method resolution order:

```

TransientVariable
cdms.avariable.AbstractVariable
cdms.cdmsobj.CdmsObj
cdms.internattr.InternalAttributesClass
PropertiedClasses.Properties.PropertiedClass
cdms.slabinterface.Slab
MA.MA.MaskedArray
    builtin .object

```

Methods defined here:

```

__init__(self, data, typecode=None, copy=0, savespace=0, mask=None, fill_value=None, grid=None,
id=None, copyaxes=1)

```

```

    createVariable (self, data, typecode=None, copy=0, savespace=
        mask=None, fill_value=None, grid=None,
        axes=None, attributes=None, id=None)

```

data is the data or a *tuple* giving the shape (not a list!)

```

__len__(self)

```

Length of first dimension

```

__repr__(self)

```

```

__str__(self)

```

```
assignValue(self, data)

astype(self, tc)
    return self as array of given type.

clone(self, copyData=1)
    clone (self, copyData=1)
    Return a copy of self as a transient variable.
    If copyData is 1 (default), make a separate copy of the data.

copyAxis(self, n, axis)
    Set n axis of self to a copy of axis. (0-based index)
    Invalidates grid.

copyDomain(self, other)
    Set the axes and grid by copying variable other.

copydimension(self, idim, other, jdim)
    Set idim dimension of self to variable other's jdim'th
    This is for old cu compatibility. Use copyAxis for new code.

expertSlice(self, slicelist)

getAxis(self, n)

getDomain(self)

getGrid(self)

getValue(self, squeeze=1)

initDomain(self, axes, copyaxes=1)

isEncoded(self)
    Transient variables are not encoded

setAxis(self, n, axis, savegrid=0)
    Set n axis of self to a copy of axis. (0-based index)

setAxisList(self, axislist)
    Set the axes to axislist.

setMaskFromGridMask(self, mask, gridindices)
    Set the mask for self, given a grid mask and the variable dom
    indices corresponding to the grid dimensions.

setMissing(self, value)
    Set missing value attribute and fill value

set_fill_value(self, value)
    Set missing value attribute and fill value
```

`setdimattribute`(self, dim, field, value)
Set the attribute named field from the dim'th dimension.

`typecode`(self)

Data and other attributes defined here:

`count = 0`

Methods inherited from [cdms.avariable.AbstractVariable](#):

`__abs__(self)`

`__add__(self, other)`

`__array__(self, t=None)`

`__call__(self, *args, **kwargs)`
Selection of a subregion using selectors

`__div__(self, other)`

`__eq__(self, other)`

`__ge__(self, other)`

`__getitem__(self, key)`

`__getslice__(self, low, high)`

`__gt__(self, other)`

`__iadd__(self, other)`
Add other to self in place.

`__idiv__(self, other)`
Divide self by other in place.

`__imul__(self, other)`
Multiply self by other in place.

`__isub__(self, other)`
Subtract other from self in place.

`__le__(self, other)`

`__lshift__(self, n)`

`__lt__(self, other)`

`__mul__(self, other)`

```

ne(self, other)

neg(self)

pow(self, other, third=None)

radd = add(self, other)

rdiv(self, other)

rmul = mul(self, other)

rshift(self, n)

rsub(self, other)

sqrt(self)

sub(self, other)

crossSectionRegrid(self, newLevel, newLatitude, missing=None, order=None, method='log')
    Return the variable regridded to new pressure levels and latitudes.
    The variable should be a function of lat, level, and (optional)
    <newLevel> is an axis of the result pressure levels.
    <newLatitude> is an axis of latitude values.
    <method> is optional, either "log" to interpolate in the log
        or "linear" for linear interpolation.
    <missing> and <order> are as for regrid.CrossSectionRegridder

decode(self, ar)
    Decode compressed data. ar is a masked array, scalar, or MA masked array.

generateGridkey(self, convention, vardict)
    generateGridkey(): Determine if the variable is gridded,
    and generate ((latname, lonname, order, maskname, class), latname)
    or (None, None, None) if not gridded. vardict is the variable dictionary.

generateRectGridkey(self, lat, lon)
    generateRectGridkey(): Determine if the variable is gridded,
    and generate (latname, lonname, order, maskname, class) if gridded
    or None if not gridded

getAxisIds(self)
    Get a list of axis identifiers

getAxisIndex(self, axis_spec)
    Return the index of the axis specified by axis_spec.
    Argument axis_spec and be as for axisMatches
    Return -1 if no match.

getAxisList(self, axes=None, omit=None, order=None)

```

Get the list of axis objects;
If axes is not None, include only certain axes.
If omit is not None, omit those specified by omit.
Arguments omit or axes may be as specified in axisMatchAxis
order is an optional string determining the output order

getAxisListIndex(self, axes=None, omit=None, order=None)

Return a list of indices of axis objects;
If axes is not None, include only certain axes.
less the ones specified in omit. If axes is None,
use all axes of this variable.
Other specifications are as for axisMatchIndex.

getConvention(self)

Get the metadata convention associated with this object.

getGridIndices(self)

Return a tuple of indices corresponding to the variable grid.

getLatitude(self)

Get the first latitude dimension, or None if not found.

getLevel(self)

Get the first vertical level dimension in the domain,
or None if not found.

getLongitude(self)

Get the first longitude dimension, or None if not found.

getMissing(self, asarray=0)

Return the missing value as a scalar, or as
a Numeric array if asarray==1

getOrder(self, ids=0)

getOrder(ids=0) returns the order string, such as tzyx.

if ids == 0 (the default) for an axis that is not t,z,x,y
the order string will contain a '-' in that location.
The result string will be of the same length as the number
of axes. This makes it easy to loop over the dimensions.

if ids == 1 those axes will be represented in the order
string as (id) where id is that axis' id. The result will
be suitable for passing to order2index to get the
corresponding axes, and to orderparse for dividing up into
components.

getRegion(self, *specs, **keys)

getRegion

Read a region of data. A region is an n-dimensional
rectangular region specified in coordinate space.

'slices' is an argument list, each item of which has one of t

- `x`, where `x` is a scalar
Map the scalar to the index of the closest coordinate value
- `(x,y)`
Map the half-open coordinate interval $[x,y)$ to index interval
- `(x,y,'cc')`
Map the closed interval $[x,y]$ to index interval. Other options: '`oc`' (open on the left), and '`co`' (open on the right, the default)
- `(x,y,'co',cycle)`
Map the coordinate interval with wraparound. If no cycle is specified, a wraparound will occur iff `axis.isCircular()` is true.
NOTE: Only one dimension may be wrapped.
- Ellipsis
Represents the full range of all dimensions bracketed by nested ellipses
- `::` or None
Represents the full range of one dimension.

For example, suppose the variable domain is `(time,level,lat,lon)`:

```
getRegion((10,20),850,Ellipsis,(-180,180))
```

retrieves:

- all times `t` such that $10 \leq t < 20$.
- level 850
- all values of all dimensions between level and lon (namely time and lat).
- longitudes `x` such that $-180 \leq x < 180$. This will be wrapped around if `lon.topology == 'linear'`

`getSlice(self, *specs, **keys)`

`x.getSlice` takes arguments of the following forms and produces a return array. The keyword argument `squeeze` determines whether or not the shape of the returned array contains dimensions whose length is 1; by default this argument is 1, and such dimensions are 'squeezed out'.

There can be zero or more positional arguments, each of the following forms:

- a single integer `n`, meaning `slice(n, n+1)`
- an instance of the slice class
- a tuple, which will be used as arguments to create a slice
- None or `::`, which means a slice covering that entire dimension
- Ellipsis (...), which means to fill the slice list with enough ellipses leaving only enough room at the end for the remaining positional arguments

There can be keyword arguments of the form `key = value`, where `key` can be one of the names '`time`', '`level`', '`latitude`', or '`longitude`'. The corresponding value can be any of (a)-(d) above.

There must be no conflict between the positional arguments and the keywords.

In (a)-(c) negative numbers are treated as offsets from the end of that dimension, as in normal Python indexing.

`getTime(self)`

Get the first time dimension, or None if not found

isAbstractCoordinate(self)

pressureRegrid(self, newLevel, missing=None, order=None, method='log')

Return the variable regridded to new pressure levels.
The variable should be a function of lat, lon, pressure, and
<newLevel> is an axis of the result pressure levels.
<method> is optional, either "log" to interpolate in the log
or "linear" for linear interpolation.
<missing> and <order> are as for regrid.PressureRegridder.

rank(self)

reg_specs2slices(self, initSpecList, force=None)

regrid(self, toGrid, missing=None, order=None, mask=None)

return self regridded to the new grid. Keyword arguments
are as for regrid.Regridder.

reorder(self, order)

return self reordered per the specification order

select = __call__(self, *args, **kwargs)

Selection of a subregion using selectors

setGrid(self, grid)

Set the variable grid

specs2slices(self, specList, force=None)

Create an equivalent list of slices from an index specification.
An index specification is a list of acceptable items, which are:
-- an integer
-- a slice instance (slice(start, stop, stride))
-- the object "unspecified"
-- the object None
-- a colon
The size of the specList must be rank()

subRegion(self, *specs, **keys)

subSlice(self, *specs, **keys)

Methods inherited from [cdms.cdmsobj.CdmsObj](#):

dump(self, path=None, format=1)

dump(self, path=None, format=1)

Dump an XML representation of this object to a file.
'path' is the result file name, None for standard output.
'format'==1 iff the file is formatted with newlines for readability.

matchPattern(self, pattern, attribute, tag)

```

# Match a pattern in a string-valued attribute. If attribute
# search all string attributes. If tag is not None, it must m

matchone(self, pattern, atname)
    # Return true iff the attribute with name atname is a string
    # attribute which matches the compiled regular expression pat
    # if atname is None and pattern matches at least one string
    # attribute. Return false if the attribute is not found or is n

searchPattern(self, pattern, attribute, tag)
    # Search for a pattern in a string-valued attribute. If attri
    # search all string attributes. If tag is not None, it must m

searchPredicate(self, predicate, tag)
    # Apply a truth-valued predicate. Return a list containing a
    # if the predicate is true and either tag is None or matches
    # If the predicate returns false, return an empty list

searchone(self, pattern, atname)
    Return true iff the attribute with name atname is a string
    attribute which contains the compiled regular expression patt
    if atname is None and pattern matches at least one string
    attribute. Return false if the attribute is not found or is n
    a string.

```

Methods inherited from [cdms.internattr.InternalAttributesClass](#):

```

is_internal_attribute(self, name)
    is internal attribute(name) is true if name is internal.

replace_external_attributes(self, newAttributes)
    replace external attributes(newAttributes)
    Replace the external attributes with dictionary newAttributes

```

Methods inherited from [PropertiedClasses.Properties.PropertiedClass](#):

```

__delattr__(self, name)

__getattr__(self, name)

__setattr__(self, name, value)

get_property_d(self, name)
    Return the 'del' property handler for name that self uses.
    Returns None if no handler.

get_property_g(self, name)
    Return the 'get' property handler for name that self uses.
    Returns None if no handler.

get_property_s(self, name)

```

Return the 'set' property handler for name that self uses.
Returns None if no handler.

***set_property*(self, name, actg=None, acts=None, actd=None, nowrite=None, nodelete=None)**

Set attribute handlers for name to methods actg, acts, actd
None means no change for that action.
nowrite = 1 prevents setting this attribute.
nowrite defaults to 0.
nodelete = 1 prevents deleting this attribute.
nodelete defaults to 1 unless actd given.
if nowrite and nodelete is None: nodelete = 1

Methods inherited from [cdms.slabinterface.Slab](#):

***createattribute*(self, name, value)**

Create an attribute and set its name to value.

***deleteattribute*(self, name)**

Delete the named attribute.

***getattribute*(self, name)**

Get the attribute name.

***getdimattribute*(self, dim, field)**

Get the attribute named field from the dim'th dimension.
For bounds returns the old cu one-dimensional version.

***info*(self, flag=None, device=None)**

Write info about slab; include dimension values and weights if

***listall*(self, all=None)**

Get list of info about this slab.

***listattributes*(self)**

Return a list of attribute names.

***listdimattributes*(self, dim)**

List the legal axis field names.

***listdimnames*(self)**

Return a list of the names of the dimensions.

***setattribute*(self, name, value)**

Set the attribute name to value.

***showdim*(self)**

Show the dimension attributes and values.

Data and other attributes inherited from [cdms.slabinterface.Slab](#):

***std_slab_atts* = ['filename', 'missing_value', 'comments', 'grid_name', 'grid_type', 'time_statistic', 'lo**

Methods inherited from [MA.MA.MaskedArray](#):

__and__(self, other)
Return bitwise_and

__float__(self)
Convert self to float.

__floordiv__(self, other)
Return divide(self, other)

__int__(self)
Convert self to int.

__mod__(self, other)
Return remainder(self, other)

__or__(self, other)
Return bitwise_or

__pos__(self)
Return array(self)

__rand__ = **__and__**(self, other)
Return bitwise_and

__rfloordiv__(self, other)
Return divide(other, self)

__rmod__(self, other)
Return remainder(other, self)

__ror__ = **__or__**(self, other)
Return bitwise_or

__rtruediv__(self, other)
Return divide(other, self)

__rxor__ = **__xor__**(self, other)
Return bitwise_xor

__setitem__(self, index, value)
Set item described by index. If value is masked, mask those locations.

__setslice__(self, i, j, value)
Set slice i:j; if value is masked, mask those locations.

__truediv__(self, other)
Return divide(self, other)

__xor__(self, other)

```
    Return bitwise_xor

byte_swapped(self)
    Returns the raw data field, byte_swapped. Included for consistency
    with Numeric but doesn't make sense in this context.

compressed(self)
    A 1-D array of all the non-masked data.

dot(self, other)
    s.dot(other) = innerproduct(s, other)

fill_value(self)
    Get the current fill value.

filled(self, fill_value=None)
    A Numeric array with masked values filled. If fill_value is None
    use fill_value().

    If mask is None, copy data only if not contiguous.
    Result is always a contiguous, Numeric array.

ids(self)
    Return the ids of the data and mask areas

iscontiguous(self)
    Is the data contiguous?

itemsize(self)
    Item size of each data item.

mask(self)
    Return the data mask, or None. Result contiguous.

outer(self, other)
    s.outer(other) = outerproduct(s, other)

put(self, values)
    Set the non-masked entries of self to filled(values).
    No change to mask

putmask(self, values)
    Set the masked entries of self to filled(values).
    Mask changed to None.

raw_data(self)
    The raw data; portions may be meaningless.
    May be noncontiguous. Expert use only.

raw_mask(self)
    The raw mask; portions may be meaningless.
    May be noncontiguous. Expert use only.
```

```

savespace(self, value)
    Set the spacesaver attribute to value

size(self, axis=None)
    Number of elements in array, or in a particular axis.

spacesaver(self)
    spacesaver\(\) queries the spacesaver attribute.

tolist(self, fill_value=None)
    Convert to list

tostring(self, fill_value=None)
    Convert to string

unmask(self)
    Replace the mask by None if possible.

unshare_mask(self)
    If currently sharing mask, make a copy.

```

Properties inherited from [MA.MA.MaskedArray](#):

flat

Access array in flat form.
`get">get = _get_flat(self)`
 Calculate the flat value.
`set">set = _set_flat(self, value)`
 `x.flat = value`

imag

Access the imaginary part of the array
`get">get = _get_imaginary(self)`
 Get the imaginary part of a complex array.
`set">set = _set_imaginary(self, value)`
 `x.imaginary = value`

imaginary

Access the imaginary part of the array
`get">get = _get_imaginary(self)`
 Get the imaginary part of a complex array.
`set">set = _set_imaginary(self, value)`
 `x.imaginary = value`

real

Access the real part of the array
`get">get = _get_real(self)`
 Get the real part of a complex array.
`set">set = _set_real(self, value)`
 `x.real = value`

shape

```
tuple giving the shape of the array
get">get = _get_shape(self)
    Return the current shape.
set">set = _set_shape(self, newshape)
    Set the array's shape.
```

Data and other attributes inherited from [MA.MA.MaskedArray](#):

```
__dict__ = <dictproxy object>
    dictionary for instance variables (if defined)

__weakref__ = <attribute '__weakref__' of 'MaskedArray' objects>
    list of weak references to the object (if defined)

handler_cache_key = 'MA.MaskedArray'
```

Functions

asVariable(s, writeable=1)

Returns s if s is a Variable; if writeable is 1, return s if s is a [TransientVariable](#). If s is not a variable of the desired type, attempt to make it so and return that. If we fail raise CDMSError

id_builtin = id(...)

id(object) -> integer

Return the identity of an object. This is guaranteed to be unique simultaneously existing objects. (Hint: it's the object's memory

isVariable(s)

Is s a variable?